Haskell Notes

**Sample Problems**

* prefix :: [a] -> [a] takes everything but the last element of the array
* dropn :: Int -> [a] -> [a] takes an int and a list and returns another list with the first n elements dropped
  + 2 -> [abcd] -> [cd]
* member :: Eq a => a -> [a] -> Bool is an item that we are looking at in the list
* mymap :: (a->b) -> [a] -> [b] takes another function as input and a list and returns a list
  + Sqrt -> [1,2,3,4,5] -> [1, sqrt 2, …]
* find :: Eq a => a -> [a] -> [Int] takes an item and a list of similar type of items and returns the a list of the position of the item in the list
* evenSum :: Integral a => [a] -> a if we have a list of integral types, finds the sum of the even values of that list
* fib/fibTuple/fibResult/fastFib deals with the Fibonacci sequence

**Prefix :: [a] -> [a]**

* Returns all but the last element of the array
* Base Case: empty list and lists with element
  + prefix of empty list is not defined (not a base case)
  + prefix of single element list is an empty list (becomes the base case)
* General case: arbitrary n-long list
  + prefix of list of length n is attach the first element in the list to the front of the prefix of the tail
  + based on prefix of list of length n – 1
* Examples of prefix



**dropn :: Int -> [a] -> [a]**

* Returns the original list with the first elements removed
* Base Case: smallest possible n and smallest list
  + Smallest possible n = 0
  + Smallest list = empty list
* General Cases: arbitrary n-long list (n>0, len(list) > 0)
  + Dropping n items from non-empty list should be thought recursively
  + Drop n-1 elements from tail of the list
* Examples of dropn

**member :: EQ a => a -> [a] -> bool**

* Determines if an element is in a list
* Base Case: empty list
  + Empty list is the easiest false case
  + Arbitrary long list while searching for the first item of the list is the easiest true case
* General Case: need either getting to the item we are searching for or the end of the list
  + Getting to the item returns true
  + Reaching the end of the list should default to false
* Examples



**mymap :: (a->b) -> [a] -> [b]**

* Base case: empty list
  + Applying a function to an empty list will turn into an empty list
* General case: applying a function to every element in a list
  + Apply function not each of n elements
* Examples

**find :: eq a => a -> [a] -> [Int]**

* takes an item from a list of the same type and returns the index of that element
* Base case: focus on list
  + A list that doesn’t have that element returns an empty list
  + If the thing you are looking for is at the head of the list, return 0
  + If we know that we are not going to find the item, returns empty list
  + “Not found” means that the item is not in the list
* General case: looking for items that are not in the head of the list
  + Need a way to distinguish and keep track of the indexes

****

**evenSum :: Integral a => [a] -> a**

* Base Case: consider the smallest list
  + Lists that have no even numbers is 0
* General Case: consider the head is even or odd
  + To determine if even or odd, we can use modular division



**fib**

* Base cases: f(0) = 0, f(1) = 0
* General cases f(n) = f(n-1) + f(n-2)
* Doing this will take too much time

**fibTuple :: (Integer, Integer, Integer) -> (Integer, Integer,Integer)**



**Data Types in Haskell**

* Built in types: int, bool, char
* Lists: takes a data types and creates a list (can create nested lists)
* Can define custom types: geometry
  + data Shape = Circle Float Float Float | Rectangle Float Float Float Float
  + surface :: Shape -> Float
* Type synonyms